区間表現からMPQ-tree を効率よく構成するアルゴリズム(計算機科学の理論とその応用)

AUTHOR(S):

斎藤, 寿樹; 清見, 礼; 上原, 隆平

CITATION:

斎藤, 寿樹 ...[et al]. 区間表現からMPQ-tree を効率よく構成するアルゴリズム(計算機科学の理論とその応用). 数理解析研究所講究録 2007, 1554: 93-100

ISSUE DATE:

2007-05

URL:

http://hdl.handle.net/2433/80967

数理解析研究所講究録
第1554巻 2007年 93-100

区間表現から MPQ-tree を効率よく構成するアルゴリズム

斎藤 寿樹　清見 礼　上原 隆平
北陸先端科学技術大学院大学 情報科学研究科

1 はじめに

区間グラフは1950年代の後半に数学者のHajösと，分子生物学者のBenzerとが独立に考えたグラフクラスである [1]．あるグラフ $G=(V,E)$ ($|V|=n, |E|=m$) が区間グラフであるとは，V の各頂点を数直線上の区間に対応付けることができ，V の頂点が隣接することの必要十分条件が，対応する区間が重なりを持つときである．この区間の集合を区間グラフの区間表現という．区間グラフは一つの区間を時間，温度などと考えることにより，様々な応用がある．特に，バイオインフォマティクスにおいて，DNA の断片を一つの区間と考えることができる．このような区間グラフの実際の応用では，多くの場合区間グラフは区間表現が入力として与えられることが想定される．

MPQ-tree は区間グラフの認識のために考案されたデータ構造で，PQ-tree を拡張して得られる [2]．MPQ-tree は区間グラフの同型性の判定に用いることができる．またそれをもとに高速に複数の区間表現を作ることができ，$O(n)$ 領域で表現できるコンパクトな構造である．このように MPQ-tree は区間グラフの応用に有用なデータ構造である．

既存のアルゴリズムを用いて，区間表現を入力として MPQ-tree を構成する手順を示す．Korte と Möring の論文 [2] では，2通りの MPQ-tree の構成法を示しているが，どちらも入力はグラフである．したがって，グラフ $G=(V,E)$ から MPQ-tree を構成するには，本質的に $O(n+m)$ 領域と $O(n+m)$ 時間かかる．また，どちらも多くの場合分けが必要であり，線形時間といっても非常に複雑なアルゴリズムである．

本論文では，区間表現を入力として与え，MPQ-tree を直接構成する．このアルゴリズムはグラフ表現を用いず，$O(n)$ 時間 $O(n)$ 領域で動作する．また，スタック操作を中心とした単純なアルゴリズムで，実装も容易である．

本アルゴリズムの概要は次の通りである．(1) 入力された区間表現から冗長性を含まず，番号付けもある規則を満たすコンパクトな区間表現を求める．(2) コンパクトな区間表現から MPQ-tree を構成する．

2 準備

2.1 区間グラフ

グラフ $G=(V,E)$ に対し，数直線上の区間の集合 $\{I_v|v\in V\}$ が存在し，以下を満たすとき，G は区間グラフであるという [1]．すなわち

$\forall v_1, v_2 \in V, (v_1,v_2)\in E \Leftrightarrow I_{v_1}$ と I_{v_2} は共通部分を持つ．

このような区間の集合 $\{I_v\}_{v\in V}$ を G の区間表現という．本論文において，それぞれの区間は閉区間とする．つまり，区間 I_v の右端点と区間 I_u の左端点が数直線上の同じ値に存在するとき，2つの区間は共通部分をもつ．区間グラフと対応する区間表現の例を図1に示す．

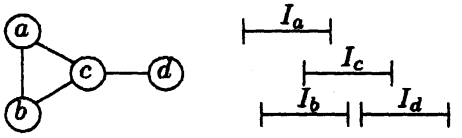

図 1: 区間グラフと対応する区間表現

2.2 区間表現

1章で述べたように，現実の問題では実際のデータを元にした区間表現（時間軸や DNA の文字列など）が入力として与えられることが多い．

本論文では次のような区間表現を入力として扱う．

1. 各区間の端点は数直線上の整数点に存在する.
2. 複数の端点が同じ整数点上に存在しない.

この区間表現には冗長性が存在するため，この区間表現を**冗長な区間表現**と呼ぶ．図 2(a) は冗長な区間表現の例である．

区間 x に対し，次の値を定義しておく．

1. $l(x)$：左の端点が存在する数直線上の値
2. $r(x)$：右の端点が存在する数直線上の値
3. $length(x)$：区間 x の長さ $(r(x) - l(x))$

先ほど述べたように，入力として与える区間表現には冗長性が存在するため，処理が複雑になってしまう．そこで，冗長性のない区間表現に変換する．

ここで**コンパクトな区間表現**を定義する．整数点 i を含む区間の集合を $N[i]$ とする．それぞれの区間の端点が 1 から始まる連続した整数点上に存在し，

$$N[i]\backslash N[i+1] \neq \phi \text{ かつ } N[i+1]\backslash N[i] \neq \phi$$

が各整数点 i について成り立つとき，この区間表現はコンパクトな区間表現であるという．図 2(b) は (a) に対応したコンパクトな区間表現である．

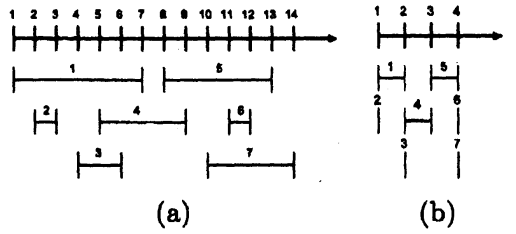

図 2: 冗長な区間表現と対応するコンパクトな区間表現

区間の番号に対して，一定の規則を仮定できると，構成アルゴリズムを単純化できる．そこで，次の規則で区間の番号付けを行う．

1. $l(x) < l(y)$ ならば $x < y$.
2. $l(x) = l(y)$ かつ $length(x) > length(y)$ ならば $x < y$.

このような順序に番号を付け替えたコンパクトな区間表現を**左端点優先の長さ順序**のコンパクトな区間表現ということにする．この左端点優先の長さ順序のコンパクトな区間表現を用いると，*MPQ*-tree の構成を単純にすることができる．

ここで，以下を定義する．相異なる 2 つの区間 x, y の端点が

$$l(x) < l(y) \leq r(x) < r(y) \text{ または } l(y) < l(x) \leq r(y) < r(x)$$

のとき，x と y は**部分交差**しているという．図 3(a)，(b) の区間 x と y は部分交差している．

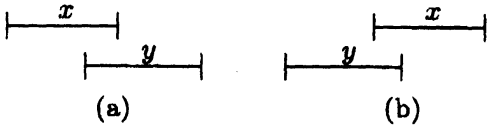

図 3: 部分交差

2.3 *PQ*-tree と *MPQ*-tree

PQ-tree は区間グラフを認識するために導入されたデータ構造である [4]．*PQ*-tree は与えられたグラフ $G = (V, E)$ を表すためのデータ構造である．*PQ*-tree は P ノードと Q ノードの 2 種類の内部ノードを持つ順序木で，以下を満たすものとして定義される．(1)Q ノードの子の数は 3 以上である．また，葉はラベルを持つ．(2) 区間グラフ G に対応する *PQ*-tree において，T の葉のラベルは G の極大クリークと 1 対 1 に対応している．(3)*PQ*-tree T に対して，次の 2 つの操作を有限回当てはめることで，他の *PQ*-tree T' を得たとき，T' も G に対応する *PQ*-tree である．さらに，G に対応する任意の *PQ*-tree T'' はこの操作で T から得られる．

1. P ノードの子の順序を任意に入れ換える.
2. Q ノードの子の左右の順序を逆にする.

MPQ-tree(Modified *PQ*-tree) は *PQ*-tree を拡張したデータ構造である [2]．*MPQ*-tree を用いることで，入力の線形時間で区間グラフの同型性判定を行うことができる [2][5]．また *MPQ*-tree を用いて与えられた区間グラフに対応するすべての区間表現を作り出すことができる．

MPQ-tree は *PQ*-tree から次のように構成されるデータ構造である．T を区間グラフ $G = (V, E)$ に対応する *PQ*-tree とする．このとき，T の葉に

はGの極大クリークのラベルが存在する．*MPQ*-tree T^*はTの各ノードに対し，次の規則で頂点の集合を割り当てることで構成される．

1. Pノード$\hat{P}$に関して，$\hat{P}$は，Tにおける$\hat{P}$のすべての部分木に存在する極大クリークに含まれ，かつTの部分木以外の極大クリークに存在しないすべての頂点のラベルを持つ．
2. Qノード$\hat{Q}$に関して，Tにおける$\hat{Q}$の子供を$Q_1,\ldots,Q_k$とし$(k \geq 3)$，Q_iを根としたTの部分木をT_iとする．それぞれのQ_iに対して，$\hat{Q}$にセクションと呼ばれる集合S_iを割り当てる．セクションS_iは，T_iの極大クリークと他の部分木T_jに含まれ，かつ$\hat{Q}$ではない他のTの部分木に存在している極大クリークに含まれていないすべての頂点のラベルを持つ．

このようなラベル，セクションの割り当ては必ず存在し，*PQ*-tree と *MPQ*-tree は1対1に対応することが容易に確認できる．

図4(a)は区間グラフである．(b)は(a)に対応する*PQ*-treeであり，(c)は(a)に対応する*MPQ*-treeである．

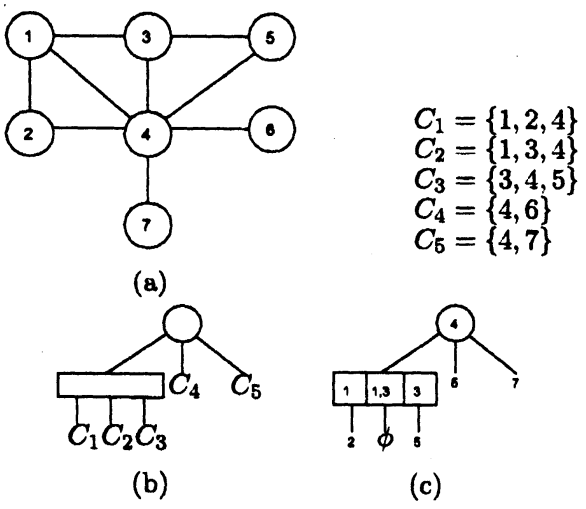

図 4: グラフとグラフに対応する *PQ*-tree と *MPQ*-tree

MPQ-tree には次の性質がある．

定理 1. [2][3] T^*を区間グラフ$G=(V,E)$から与えられた*MPQ*-treeとする．

1. T^*は$O(n+m)$時間で得ることができ，$O(n)$領域で表現できる．
2. Gのそれぞれの極大クリークはT^*の根から葉までのパス上に存在する頂点ラベルの集合の1つと一致する．
3. T^*において，Gの頂点vは1つの葉，または1つのPノード，または1つのQノードの2つ以上連続したセクションに1回だけ現れる．
4. T^*の根は全ての極大クリークに属している頂点を含む．また，葉は単体的頂点を含む．

MPQ-tree の各セクションやそのセクションの部分木には次のような規則が存在する．

補題 2. [2][3] $\hat{Q}$を*MPQ*-treeのQノードとする．$\hat{Q}$のセクションを$S_1,\ldots,S_k$の順であるとし，またU_iを$S_i(1 \leq i \leq k)$より下の部分木に存在する頂点の集合とする．このとき，以下の性質が成り立つ．

1. $S_{i-1} \cap S_i \neq \phi$ $(2 \leq i \leq k)$,
2. $S_1 \subseteq S_2$ かつ $S_k \subseteq S_{k-1}$,
3. $U_1 \neq \phi$ かつ $U_k \neq \phi$,
4. $(S_i \cap S_{i+1}) \setminus S_1 \neq \phi$ かつ $(S_{i-1} \cap S_i) \setminus S_k \neq \phi$ $(2 \leq i \leq k-1)$,
5. $S_{i-1} \neq S_i$ $(2 \leq i \leq k)$,
6. $(S_{i-1} \cup U_{i-1}) \setminus S_i \neq \phi$ かつ $(S_i \cup U_i) \setminus S_{i-1} \neq \phi$ $(2 \leq i \leq k)$.

3 コンパクトな区間表現の構成

3.1 コンパクトな区間表現の構成

まず，冗長な区間表現のデータ構造を示す．入力として与えられる冗長な区間表現*RI*は実際のデータの各端点を順番に並べたものである．*RI*は端点が区間表現の左にあるものから順に双方向連結リストで格納されると仮定する．双方向連結リストの最初の端点を*head*(*RI*)で与える．各端点は区間の番号$(=\{1,\ldots,n\})$と端点の種類$(=\{L,R\})$の2組のデータを持つ．

次に，コンパクトな区間表現のデータ構造を示す．コンパクトな区間表現の最大の整数点を*maxCI*とする．このとき，コンパクトな区間表

現は $maxCI$ 個の要素を持つ配列 $CI[1..maxCI]$ である．添字は区間表現の整数点であり，配列の各要素はその整数点上に存在する端点の連結リストである．各端点は区間の番号と端点の種類の2組のデータを持つ．

コンパクトな区間表現をこのようなデータ構造にするとスイープがしやすくなる．スイープとはすべての整数点の端点を1通り読み込む処理である．

次のようにして，冗長な区間表現 RI をコンパクトな区間表現 CI に変換する．まず，RI の端点を左から順番に読み込んでいく．RI の連続する2つの端点が次の順に現れるとき，CI では2つの端点は同一整数点上に存在する．

1. 左の端点 → 右の端点
2. 左の端点 → 左の端点
3. 右の端点 → 右の端点

それぞれを図示すると，図5の(a)，(b)，(c)のときである．

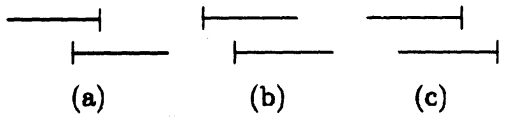

図 5: 同一整数点上に存在する端点

RI の連続する2つの端点が右の端点 → 左の端点のとき，CI では2つの端点は同じ整数点上に存在しない．図6のようなとき，x の右の端点と y の左の端点は同じ整数点上に存在しないように，整数点をシフトさせ，端点を格納する．

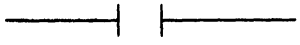

図 6: 同一整数点上に存在しない端点

3.2 左端点優先の長さ順序のコンパクトな区間表現の構成

左端点優先長さ順序のコンパクトな区間表現のデータ構造はコンパクトな区間表現と同じである．さらに，コンパクトな区間表現のデータ構造のリストの格納順の規則を与える．1つの整数点上で，まず番号の小さい順に左端点を格納する．次に番号の大きい順に右端点を格納する．

ここでコンパクトな区間表現 CI を入力とし，左端点優先の長さ順序のコンパクトな区間表現 $LLorderCI$ を出力するアルゴリズムについて説明する．このアルゴリズムは CI に対してスイープを行う．このときどの区間も左端点から読み込まれる．

スイープが $CI[i-1]$ の端点をすべて読み込んだとする．また，ここまでに読み込んだ左端点の数を h とする．$CI[i]$ を見たとき，$(x_1, L) \rightarrow (x_2, L) \rightarrow \cdots \rightarrow (x_k, L)$ の順で左の端点が格納されていたとする．(x_1, L) が $h+1$ 個目に読み込まれた左端点であるので，$LLorderCI[i]$ の左端点は $(h+1, L) \rightarrow (h+2, L) \rightarrow \cdots \rightarrow (h+k, L)$ の順番に格納される．図7はその様子である．図の(a)は CI であり，(b)は $LLorderCI$ である．

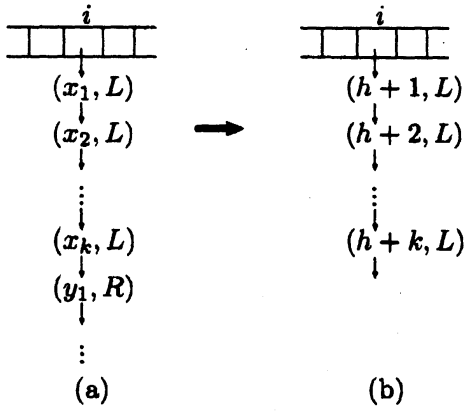

図 7: 左端点の格納

ある区間の左端点 (x_a, L) が $CI[i]$ に存在し，右端点 (x_a, R) が $CI[j]$ に存在したとする $(i \leq j)$．このとき，x_a は $h+1, \ldots, h+k$ のうちのいずれかである．左から端点を読み込んでいくので，区間の長さが短い順に番号を付けることが簡単である．$LLorderCI$ から，左端点が同じ整数点上に存在するとき，区間は長さの短い方から大きい番号を付ける．よって，$CI[i]$ に左端点が存在する区間は右端点を読み込まれた順に $h+k, h+(k-1), \ldots, h+1$ の番号を付ける．そして，x_a は $h+b$ に番号付けされるとき，$h+b$ の右端点が j に存在していたことを配列 $ExistR$ に格納しておく．スイープが終わっ

てから，$ExistR$を後ろから参照して，$LLorderCI$の区間の番号の大きい順から右端点を$LLorderCI$に格納していく．その様子を図8に示す．

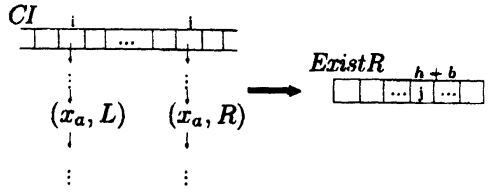

図 8: 区間の番号付け

4 MPQ-treeの構成

本章では*MPQ*-treeを構成する手順について述べる．*MPQ*-treeにはPノード，Qノード，葉が存在する．本章において，葉は子を持たないPノードとして扱う．まず，3章のアルゴリズムを用いて，入力の区間表現を左端点の長さ順序のコンパクトな区間表現$LLorderCI$に変換する．次に，$LLorderCI$を入力とし，次の2つのアルゴリズムを用いて*MPQ*-treeを構成する．

1つ目のアルゴリズムは，$LLorderCI$を入力として，それぞれの区間が属するノードとその種類(PノードかQノードか)を表す配列$LaminarA$を作成する．このアルゴリズムは図9の(a)の区間表現$LLorderCI$から(b)のノードを作成する．

2つ目のアルゴリズムで，作成した$LaminarA$と入力の$LLorderCI$を用いて，*MPQ*-treeを構成する．このアルゴリズムでは分類結果の情報を元に，それぞれのPノードやQノードの親子関係を築き，*MPQ*-treeの構造を構成する．図9の(c)は(a),(b)の情報を元にして，親子関係を築いた*MPQ*-treeである．

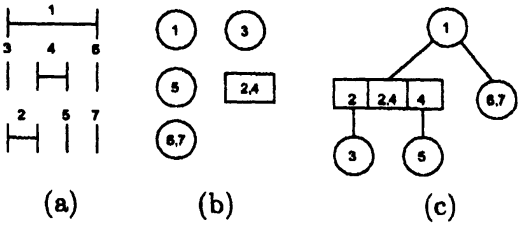

図 9: アルゴリズムの流れ

4.1節で区間をPノード，Qノード，葉に分類するアルゴリズムについて記述し，4.2節で分類されたノードの親子関係を築くアルゴリズムについて記述する．

4.1 区間を属するノードに分類するアルゴリズム

区間を各ノードに分類するアルゴリズムをアルゴリズム1に示す．このアルゴリズムの入力は左端点優先のコンパクトな区間表現$LLorderCI$であり，出力はそれぞれ区間が属するノードとその種類を表す配列$LaminarA[1..n]$である．nは入力の区間の数である．

アルゴリズム1はスイープを1回行う．スイープを行うと，任意の区間は右の端点より先に左の端点が読み込まれる．スイープの途中で，左の端点を読み込み，右の端点を読み込んでいない区間を不完全な区間という．それに対し，左と右の両方の端点が読み込まれた区間のことを完全な区間という．また，このアルゴリズム1は区間がどのノードに分類されるかがわかったとき，その都度分類を行う．そこで，不完全な区間が存在するノードのことを不完全なノードと呼び，完全な区間しか存在しないノードを完全なノードと呼ぶ．

アルゴリズム1は次の定理3の性質を用いて，同じQノードに分類される区間を見つける．

定理 3. コンパクトな区間表現において，区間xとyが部分交差しているとき，xとyの区間の番号は同じQノードに分類される．また，区間xとyがQノードに分類されるとき，$l(x) = l(z)$かつ$r(y) = r(z)$となるような区間zの番号も同じQノードに分類される．

スタックSに対して次の操作を行い，部分交差している区間を見つける．まず，スイープで区間xの左端点を読み込んだとき，$Push(S, x)$を行う．区間xの右端点を読み込んだとき，Sの先頭の要素とxを比較する．xがSに格納されている区間と部分交差しているとき，Sの先頭の要素とxは一致しない．そして，Sの先頭からxが格納されているSの要素までに現れる区間の番号はすべて

同じ Q ノード $\hat{Q}$ に分類される（図 10(a), (b)）．右端点が読み込まれ，$\hat{Q}$ が完全なノードとなるとき，$\hat{Q}$ に分類される区間の番号をすべて取り去り，$\hat{Q}$ に分類されることを $LaminarA$ に格納する（図 10(c)）．

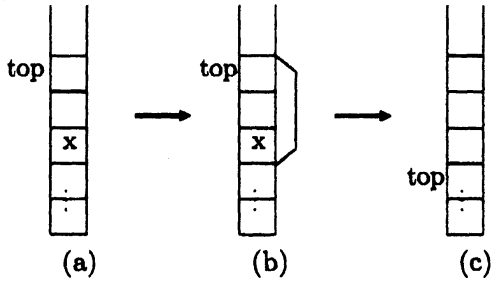

図 10: 同じ Q ノードに分類

区間 x がすでに不完全な Q ノード $\hat{Q}$ に分類されているとする．ここで，区間 x の右端点が読み込まれたとき，スタックの先頭から $\hat{Q}$ に含まれる区間を同一の Q ノードに分類する．このとき，スタックの先頭と x の間に $\hat{Q}$ 以外の不完全な Q ノード $\hat{Q}_i$ が存在したとすると，$\hat{Q}_i$ と $\hat{Q}$ はマージされ，$\hat{Q}_i$ に分類されている区間は $\hat{Q}$ と同じ Q ノードに分類される．

マージの回数を求めるため，マージ関係を表す木 T_m を用いる（図 11）．T_m の葉を区間と考えると，葉の数は n 個である．1 回のマージ操作が 1 つの内部ノードに対応させると，内部ノードの数は高々 n であるので，マージを行う回数は高々 n 回である．また，2 つの不完全な Q ノードをマージするとき，2 つのノードのスタックでの添字の範囲を格納しておくことで，マージにかかる時間は $O(1)$ で可能である．よって，全体のマージは $O(n)$ 時間かかる．

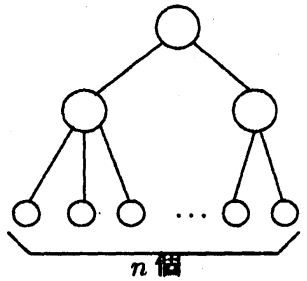

図 11: マージ関係を表す木

Algorithm 1: 区間を各ノードに分類するアルゴリズム

Input: 左端点優先の長さ順序のコンパクトな区間表現 $LLorderCI$
Output: 属するノードと種類を表す配列 $LaminarA$

```
1  for i = 1 to maxCI do
2      e ← head(LLorderCI[i]);
3      while e ≠ NIL do
4          if kind(e) = L then
5              Push(S, num(e));
6          else
7              if top(S) ≠ num(e) または top(S) がQノードに分類 then
8                  Sの先頭から numまでが同じQノード;
9                  if 不完全なQノードが完全になる then
10                     Sから Qノードの区間を取り出す;
11                     取り出した区間を Qノードに分類;
12                     flag ← 1;
13                 end
14             else
15                 if flag ← 1 then
16                     num(e) を Qノードに分類;
17                 else
18                     Pノード作成処理;
19                 end
20                 スタックから num(e) を取り出す;
21             end
22         end
23     end
24 end
```

4.2 親子関係の構築アルゴリズム

親子関係の構築アルゴリズムをアルゴリズム 2 に示す．アルゴリズム 2 は左端点優先の長さ順序のコンパクトな区間表現 *LLorderCI* と区間の属するノードへの分類アルゴリズムで求めた配列 *LaminarA* から *MPQ*-tree を構成する．

MPQ-tree のすべてのノード N は次のようなデータを持つ．

1. ノードの種類 $kind(N)$
2. ノードの番号 $num(N)$
3. 親へのポインタ $parent(N)$

P ノード $\hat{P}$ は上に加えて次のデータを持つ．

1. 区間の集合 $set(\hat{P})$
2. 子へのポインタのリスト $Child(\hat{P})$
   特に，リストの先頭は $\hat{P}$ の末子で $head(Child(\hat{P}))$ と表す．

P ノードを図示すると図 12 のようになる．

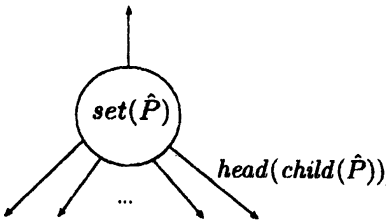

図 12: P ノード $\hat{P}$ のデータ構造

Q ノード $\hat{Q}$ のデータ構造について説明する．$\hat{Q}$ はセクション $S_1, S_2, \ldots, S_k$ で構成されており，それぞれのセクションが子を持つ．Q ノードの末子とつながっているセクションを末子セクションという．図で表すと，末子セクションは一番右に存在するセクションである．$\hat{Q}$ の末子セクションを S_k とする．$\hat{Q}$ は S_k へのポインタ $section(\hat{Q})$ を持つ．

セクション S_i は次のようなデータを持つ．

1. Q ノードへのポインタ $node(S_i)$
2. 子へのポインタ $child(S_i)$
3. 両隣りのセクションへのポインタ
   S_i の右隣りのセクション： $right(S_i)$
   S_i の左隣りのセクション： $left(S_i)$

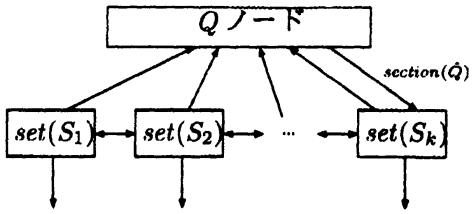

図 13: Q ノードのデータ構造

4. 端点の集合 $set(S_i)$

Q ノード $\hat{Q}$ のデータ構造は図 13 になる．

アルゴリズム 2 はスタック S を用いて親子関係を築く．S にはノードの種類と番号の組を格納する．左の端点 (x, L) を読み込んだとき，x がラベル付けされるノードを N_x とする．N_x のフラグが立っているかを調べる．フラグが立っているとき，N_x は S に格納されており，フラグが立っていないとき，N_x は S に格納されていない．フラグが立っているとき，x は S の先頭のノードにラベル付けされる．フラグが立っていないとき，N_x は S の先頭のノードの子になり，N_x を S に格納する．フラグが立っているときの例を図 14 に示し，フラグが立っていない例を図 15 に示す．

右の端点 (x, R) が読み込まれたとする．このとき，ノード N_x が完全であるとき，$POP(S)$ を行う．ノード N_x が完全ではないとき，何もしない．

以上のように，各ノードの親子関係を構築する．

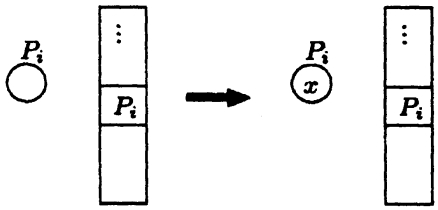

図 14: フラグが立っているときの例

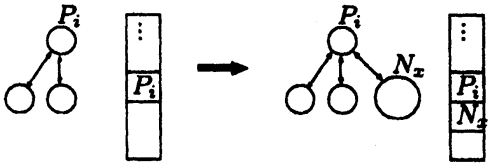

図 15: フラグが立っていないときの例

Algorithm 2: 親子関係の構築アルゴリズム

Input: 左端点優先の長さ順序のコンパクトな区間表現 *LLorderCI*, 属するノードと種類を表す配列 *LaminarA*

Output: 対応する *MPQ*-tree

```
1  for i = 1 to maxCI do
2      e ← head(LLorderCI[i]);
3      while e ≠ NIL do
4          N_x ← LaminarA[num(e)];
5          N_x に端点 e を書き込む;
6          if kind(e) = L then
7              if N_xのフラグ = 0 then
8                  スタックの先頭の子に N_x を
                   作成;
9                  PUSH(S, N_x);
10             end
11             N_x のフラグを 1 増加;
12         else
13             N_x のフラグを 1 減少;
14             if N_xのフラグ = 0 then
15                 POP(S);
16             end
17         end
18     end
19     if kind(N_x) = Q then
20         if set(Section(N_x)) が空 then
21             セクション Section(N_x) と
               left(Section(N_x)) を結合;
22         end
23         N_x に新しくセクションを作成;
24     end
25 end
```

5 おわりに

区間表現を入力として与えたとき，その区間表現に対応する *MPQ*-tree を構成する場合分けの少ない，かつ高速なアルゴリズムを提案した．

今後の課題は本質的に異なるコンパクトな区間表現の列挙が考えられる．また本論文で提案したアルゴリズムを実装して，実際に高速であることを実験的に示す．

参考文献

[1] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* ANNALS OF DISCRETE MATHEMATICS 57. ELSEVIER, 2004.

[2] N. Korte and R. H. Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM Journal on Computing*, 18(1):68–81, 1989.

[3] R. Uehara. Canonical Data Structure for Interval Probe Graphs. *Lecture Notes in Computer Science Vol. 3341*, pp. 859–870, 2004.

[4] K. S. Booth and G. S. Lueker testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using *PQ*-Tree Algorithms. *Journal of Compter and System Sciences*, 13, 335–379.

[5] G. S. Lueker and K. S. Booth A Linear Time Algorithm for Deciding Interval Graph Isomorphism. *Journal of the Association for Computing Machinery*, vol. 26, No.2, pp. 183–195, 1979.